[100] 213 revised on 2012.10.28

cemmath
The Simple is the Best

vertex, csys

```
//-------------------------------------------------------------------------
// write vertex
//-------------------------------------------------------------------------
 < x,y,z      >   // same as < x,y,z, rec > in rectangular coordinate
 < r,t,z, cyl >   // cylindrical coordinate
 < R,P,T, sph >   // spherical coordinate

#> < 1, 2, 3 > ;  < 1, 2, 3, rec > ;
 ans = <            1            2            3 >
 ans = <            1            2            3 >

// radian is default
#>  < 1, pi/3,    3, cyl > ;  cos(pi/3);  sin(pi/3);
ans = <          0.5        0.866            3 >
ans =             0.5
ans =       0.8660254

%> definition of sphereial coordinate (R,P,T)
#> R=1;;   P=pi/6;;   T=pi/3;;
#> x = R*sin(P)*cos(T);    y = R*sin(P)*sin(T);   z = R*cos(P);
 x =            0.25
 y =       0.4330127
 z =       0.8660254

#> < R, P, T, sph > ;
 ans = <          0.25         0.433         0.866 >

//-------------------------------------------------------------------------
// read vertex
//-------------------------------------------------------------------------
 v .rec         // read in rectangular coordinate
 v .cyl         // read in cylindrical coordinate
```

```
 v .sph          // read in spherical coordinate

#> < 1,1,1 > .rec ;
 ans = <            1            1            1 >

#> < 1,1,1 > .cyl ;   // r = sqrt(x^2+y^2)
 ans = <        1.414       0.7854            1 >

#> < 1,1,1 > .sph ;   // R = sqrt(x^2+y^2+z^2)
 ans = <        1.732       0.9553       0.7854 >

//---------------------------------------------------------------------
// write inside, read outside for csys cs1,cs2
//---------------------------------------------------------------------
 < a,b,c, cs1 > .cs2  // write in csys cs1, read in csys cs2

#>   < sqrt(2), pi/4, 1, cyl > ;
 ans = <            1            1            1 >

#>   < sqrt(2), pi/4, 1, cyl > .cyl ;
 ans = <        1.414       0.7854            1 >

#>   < sqrt(2), pi/4, 1, cyl > .sph ;
 ans = <        1.732       0.9553       0.7854 >

//---------------------------------------------------------------------
// assignment by a constant
//---------------------------------------------------------------------
 'vertex = double' assignment is defined as x = y = z = s.

#> v = < 1,2,3 > ;
 v = <            1            2            3 >

#> v = 5 ;
 v = <            5            5            5 >

//---------------------------------------------------------------------
// coordinate values (x,y,z), (r,t,z), (R,P,T)
//---------------------------------------------------------------------
 v .x
 v .y
```

```
 v .z

 v .r
 v .t

 v .R
 v .P
 v .T

%> rectangular coordinate ------------------------------------------------
#> v = < 1,1,1 > ;  v.x; v.y; v.z;
 v = <            1            1            1 >
 ans =               1
 ans =               1
 ans =               1

#> v.x += 1 ;  v.x; v.y; v.z;    // only x is changed
 v = <            2            1            1 >
 ans =               2
 ans =               1
 ans =               1

%> cylindrical coordinate ------------------------------------------------
#> v = < 1,1,1 > ;  v.r; v.t; v.z;
 v = <            1            1            1 >
 ans =       1.4142136
 ans =      0.78539816
 ans =               1

#> v.r += 1;  v.r; v.t; v.z;  // only r is changed
v = <        1.707        1.707            1 >
 ans =       2.4142136
 ans =      0.78539816
 ans =               1

%> spherical coordinate --------------------------------------------------
#> v = < 1,1,1 > ;  v.R; v.P; v.T;
 v = <            1            1            1 >
 ans =       1.7320508
 ans =      0.95531662
 ans =      0.78539816
```

```
#> v.P += 1;  v.R; v.P; v.T;  // only P is changed
v = <        1.135        1.135      -0.6497 >
 ans =       1.7320508
 ans =      1.95531662
 ans =      0.78539816

//---------------------------------------------------------------------
// unary operations
//---------------------------------------------------------------------
 |v|           // sqrt( v.x*v.x + v.y*v.y + v.z*v.z )
 -v            // < -x,-y,-z >

#> | <1,1,1> | ;
 ans =       1.7320508

//---------------------------------------------------------------------
// binary operations
//---------------------------------------------------------------------
 u + v
 u - v
 u * v         // dot product,       u.x*v.x + u.y*v.y + u.z*u.z
 u ^ v         // cross product,   < u.y*v.z-u.z*v.y, ... >

 v + x,        x + v
 v - x,        x - v
 v * x,        x * v
 v / x,         ...
 ...           x \ v

#>   <1,2,3> + <3,4,5> ;
 ans = <            4            6            8 >

#>   <1,2,3> * <3,4,5> ;
 ans =              26

#>   <1,2,3> ^ <3,4,5> ;    // < 2*5-3*4,   3*3-1*5,  1*4-2*3 >
 ans = <           -2            4           -2 >

//---------------------------------------------------------------------
// element-by-element binary operations
```

```
//-----------------------------------------------------------------------
 u .* v        //   < u.x*v.x, u.y*v.y, u.z*v.z >
 u ./ v        //   < u.x/v.x, u.y/v.y, u.z/v.z >
 u .\ v        //   < u.x\v.x, u.y\v.y, u.z\v.z >
 u .^ v        //   < u.x^v.x, u.y^v.y, u.z^v.z >
 v .^ s        //   < v.x^s , v.y^s , v.z^s >

 x .^ s        //   < x^v.x , y^v.y , z^v.z >

//-----------------------------------------------------------------------     //
conditional operations
//-----------------------------------------------------------------------
 u == v
 u != v
 u >  v             // true for all coordinate values
 u <  v
 u >= v
 u <= v

v == x,      x == v
v != x,      x != v
v >  x,      x >  v
v <  x,      x >  v
v >= x,      x >= v
v <= x,      x <= v

#>   <1,2,3> == <3,4,5> ;
 ans =               0

#>   <1,2,3> != <3,4,5> ;
 ans =               1

#>   <1,2,3> > <3,4,5> ;
 ans =               0

#>   <1,2,3> < <3,4,5> ;
 ans =               1

#>   <1,2,3> >= <3,4,5> ;
 ans =               0
```

```
#>   <1,2,3> <= <3,4,5> ;
 ans =                1

//-----------------------------------------------------------------------
// member functions
//-----------------------------------------------------------------------
 v .unit   // normalize

 v .xrot(t)
 v .yrot(t)
 v .zrot(t)

 v .xrotd(t)
 v .yrotd(t)
 v .zrotd(t)

v .trun(eps=1.e-30)
//  truncate x, y, z  if less than eps,  (v is changing !)
v .trun1,trun2,...,trun16
// truncate with eps=10^-1, 10^-2, ..., 10^-16,  (v is changing !)

#>  v = < 1,1,1 > ;  v.zrotd(45).trun10;
 v = <            1            1            1 >
 ans = <            0        1.414            1 >

//-----------------------------------------------------------------------
// Tensor Operations
//-----------------------------------------------------------------------
```

Vectors in physics, e.g. force, can be also described by vertices in 3D space. In Cemmath, the second-order tensor in physics is treated as a 3 x matrix.

Then, two vertices, or equivalently two vectors

```
a = < a1, a2, a3 >
b = < b1, b2, b3 >
```

can be used to define tensor product and tensor division

```
a ** b = [ a1*b1, a2*b1, a3*b1 ]
 [ a1*b2, a2*b2, a3*b2 ]
 [ a1*b3, a2*b3, a3*b3 ]
```

```
a %% b = [ a1/b1, a2/b1, a3/b1 ]
 [ a1/b2, a2/b2, a3/b2 ]
 [ a1/b3, a2/b3, a3/b3 ]
```

These operators can be utilized to calculate, for example a stress tensor is

$$\boldsymbol{\sigma} = \frac{d\mathbf{F}}{d\mathbf{A}} = \left(\frac{dF_j}{dA_j}\right) = \begin{bmatrix} \frac{dF_1}{dA_1} & \frac{dF_2}{dA_1} & \frac{dF_3}{dA_1} \\ \frac{dF_1}{dA_2} & \frac{dF_2}{dA_2} & \frac{dF_3}{dA_2} \\ \frac{dF_1}{dA_3} & \frac{dF_2}{dA_3} & \frac{dF_3}{dA_3} \end{bmatrix}$$

since we can write the above as ‘dF %% dA’.

In addition, the typical multiplication in continuum mechanics such as ‘vector*tensor’ and ‘tensor*vector’ can be implemented in Cemmath

```
‘vertex’  *  ‘matrix’
‘matrix’  *  ‘vertex’
```

which represent the following operations

$$\begin{aligned} d\mathbf{A}\cdot\boldsymbol{\sigma} &= (\,\mathbf{i}dA_1 + \mathbf{j}dA_2 + \mathbf{k}dA_3) \\ &\quad \cdot (\mathbf{ii}\sigma_{11} + \mathbf{ij}\sigma_{12} + \mathbf{ik}\sigma_{13} + \mathbf{ji}\sigma_{21} + \mathbf{jj}\sigma_{22} + \mathbf{jk}\sigma_{23} \\ &\qquad + \mathbf{ki}\sigma_{31} + \mathbf{kj}\sigma_{32} + \mathbf{kk}\sigma_{33}) \\ &= dA_1(\mathbf{i}\sigma_{11} + \mathbf{j}\sigma_{12} + \mathbf{k}\sigma_{13}) + dA_2(\mathbf{i}\sigma_{21} + \mathbf{j}\sigma_{22} + \mathbf{k}\sigma_{23}) \\ &\quad + dA_3(\mathbf{i}\sigma_{31} + \mathbf{j}\sigma_{32} + \mathbf{k}\sigma_{33}) \end{aligned}$$

$$\begin{aligned} \boldsymbol{\sigma}\cdot d\mathbf{A} &= (\mathbf{ii}\sigma_{11} + \mathbf{ij}\sigma_{12} + \mathbf{ik}\sigma_{13} + \mathbf{ji}\sigma_{21} + \mathbf{jj}\sigma_{22} + \mathbf{jk}\sigma_{23} \\ &\quad + \mathbf{ki}\sigma_{31} + \mathbf{kj}\sigma_{32} + \mathbf{kk}\sigma_{33}) \cdot (\,\mathbf{i}dA_1 + \mathbf{j}dA_2 + \mathbf{k}dA_3) \\ &= dA_1(\mathbf{i}\sigma_{11} + \mathbf{j}\sigma_{21} + \mathbf{k}\sigma_{31}) + dA_2(\mathbf{i}\sigma_{12} + \mathbf{j}\sigma_{22} + \mathbf{k}\sigma_{32}) \\ &\quad + dA_3(\mathbf{i}\sigma_{13} + \mathbf{j}\sigma_{23} + \mathbf{k}\sigma_{33}) \end{aligned}$$

An example for tensor operations is given below.

```
#>    i = < 1,0,0 >;;   j = < 0,1,0 >;;   k = < 0,0,1 >;;
```

```
#>  va = <1,2,3>; vb = <3,4,5>;
 va = <            1            2            3 >
 vb = <            3            4            5 >

#>  C = va ** vb ;
 C =
 [            3            6            9 ]
 [            4            8           12 ]
 [            5           10           15 ]

#>  D = va %% vb ;
 D =
 [     0.333333     0.666667            1 ]
 [         0.25          0.5         0.75 ]
 [          0.2          0.4          0.6 ]



#> i * D;  j * D;  k * D;
 ans = <       0.3333       0.6667            1 >
 ans = <         0.25          0.5         0.75 >
 ans = <          0.2          0.4          0.6 >

#> D * i;  D * j;  D * k;
 ans = <       0.3333         0.25          0.2 >
 ans = <       0.6667          0.5          0.4 >
 ans = <            1         0.75          0.6 >

//-------------------------------------------------------------------------
// vertex Array
//-------------------------------------------------------------------------
```

An array of vertices is defined according to the following syntax

```
vertex A[7],B[21], ... ;
```

where A and B are the names of arrays. The dimension of an array can be a variable of type 'double'.

```
#> n=10;  vertex V[n];
 n =            10
```

```
#> for.i(0,4) V[i] = < i, i+1, i+2 > ;;

#> V;
 V = vertex [10]
 [0] = <           0           1           2 >
 [1] = <           1           2           3 >
 [2] = <           2           3           4 >
 [3] = <           3           4           5 >
 [4] = <           4           5           6 >
 [5] = <           0           0           0 >
 [6] = <           0           0           0 >
 [7] = <           0           0           0 >
 [8] = <           0           0           0 >
 [9] = <           0           0           0 >

//-------------------------------------------------------------------------
//  Output format
//-------------------------------------------------------------------------
 The screen output can be controlled by

vertex.format(string);  // default is "<%9.3g %9.3g %9.3g >"

#> vertex.format(" %g  %12.3e (%12.4f)");   < 1.2, 1.3, 1.4 > ;
 ans =  1.2    1.300e+000 (      1.4000)

#> vertex.format("%9.3g %9.3g ");  < 1.2, 1.3, 1.4 > ;
 ans =       1.2       1.3

#> vertex.format("< %9.3g %9.3g %9.3g >");  < 1.2, 1.3, 1.4 > ;
 ans = <       1.2       1.3       1.4 >



//=================================================

A coordinate system is defined by a origin vertex and three direction cosine
vertex

 .org  // origin
 .dir  // direction cosine
```

```
//-----------------------------------------------------------------------
// fundamental coordinates
//-----------------------------------------------------------------------
 csys.rec      // rectangular
 csys.cyl      // cylindrical
 csys.sph      // spherical

For example,

#> cs1 = csys.rec ;

results in

cs1 = 'rec' local csys
 org = <           0           0           0 >
 dir = [           1           0           0 ]
 [           0           1           0 ]
 [           0           0           1 ]

#> cs1.org ;
 ans = <           0           0           0 >

#> cs1.dir ;
 ans =
 [           1           0           0 ]
 [           0           1           0 ]
 [           0           0           1 ]

//-----------------------------------------------------------------------
// angle system
//-----------------------------------------------------------------------
 csys.deg      // angles in degree
 csys.rad      // angles in radian

#> csys.deg; <  1,  45,  0, cyl > ;
 ans = <      0.7071      0.7071           0 >

#> csys.rad; < 1,  pi/4,  0, cyl > ;
 ans = <      0.7071      0.7071           0 >

//-----------------------------------------------------------------------
```

```
// angle range
//-------------------------------------------------------------------------
 csys.pass180;       // pass through -x coordinate, 0 <= t < 2*pi
 csys.pass0;         // pass through +x coordinate, -pi < t <= pi

#> csys.pass180;  < 1, -1, 0 >.cyl ;
 ans = <         1.414         5.498             0 >

#> csys.pass0;    < 1, -1, 0 >.cyl ;
 ans = <         1.414       -0.7854             0 >

//-------------------------------------------------------------------------
// local coordinate system
//-------------------------------------------------------------------------
```

Let vo is the origin, and u,v are two non-parallel vertices. Then, a local coordinate system can be defined as

```
cs = csys.cyl ( vo, u, v ) ;
```

where cylindrical coordinate is chosen. Direction cosines are obtained by

```
u1 = u .unit
u3 = (u ^ v).unit
u2 = (u3 ^ u1).unit

#> cs1 = csys.cyl( <3, 1>,  <1, 1>, <-1, 2>);

cs1 = 'cyl' local csys
 org = <             3             1             0 >
 dir = [      0.707107     -0.707107             0 ]
 [       0.707107      0.707107            -0 ]
 [              0             0             1 ]
```

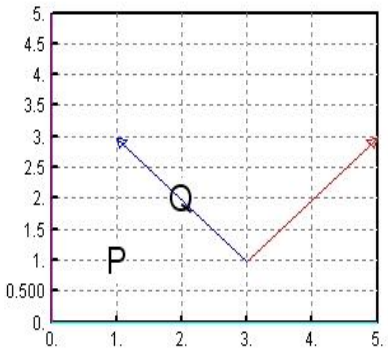

Figure 1 A local coordinate system

```
#> <1, 1, 0>.cs1 ;              // <2, pi*3/4, 0>
 ans = <            2        2.356            0 >

#> <2, 2, 0>.cs1 ;              // <1.414, pi/2, 0>
 ans = <        1.414        1.571            0 >

#> <sqrt(2), pi/2, 0, cs1> ; // <2, 2, 0>
 ans = <            2            2            0 >

//-------------------------------------------------------------------------
// change csys origin and direction
//-------------------------------------------------------------------------
 If cs is a csys object,

cs.org = vertex
cs.dir = matrix of 3 x 3 dimension

//-------------------------------------------------------------------------
// csys array
//-------------------------------------------------------------------------
 An array of  csys  is defined according to the following syntax

#> csys cs[7], cs1[21];
```

where cs and cs1 are the names of arrays. The dimension of an array can be a variable of type 'double'.

```
#> n=4;  csys cs[n];
 n =              4

#> for.i(0,3) cs[i] = csys.cyl( <i,i,i>, <1>,<0,1>);;

#> cs;
 cs = csys [4]
 [0] = 'cyl' local csys
 org = <           0           0           0 >
 dir = [           1           0           0 ]
 [           0           1           0 ]
 [           0           0           1 ]
 [1] = 'cyl' local csys
 org = <           1           1           1 >
 dir = [           1           0           0 ]
 [           0           1           0 ]
 [           0           0           1 ]
 [2] = 'cyl' local csys
 org = <           2           2           2 >
 dir = [           1           0           0 ]
 [           0           1           0 ]
 [           0           0           1 ]
 [3] = 'cyl' local csys
 org = <           3           3           3 >
 dir = [           1           0           0 ]
 [           0           1           0 ]
 [           0           0           1 ]

//-----------------------------------------------------------------
//  end of file
//-----------------------------------------------------------------
```